Journal of Computer and System Sciences 78 (2012) 1185–1203

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Personalized multi-user view and content synchronization and retrieval in real-time mobile social software applications

Haifeng Shen *, Mark Reilly

School of Computer Science, Engineering and Mathematics, Flinders University, Adelaide, Australia

ARTICLE INFO

Article history:
Received 29 April 2011
Received in revised form 22 July 2011
Accepted 17 October 2011
Available online 3 November 2011

Keywords:
Mobile device
Social software
Interface
Synchronization
Compression
Operational transformation
Operational merging
Retrieval

ABSTRACT

Past years have witnessed the rapid growth of computer-based social software. Despite the increasing popularity of mobile devices, the choices of social software on these devices are still limited to non-real-time email and social media systems. Real-time social software on mobile devices is virtually non-existent due to the device characteristics such as small screen real estate, limited battery talk time, scarce network resources, and inherent need for personalization, which present challenges to the design and implementation of effective and useful real-time mobile social software. In this article, we present a technical solution to these challenges using a smartphone-based real-time collaborative note-taking system as an example. The solution allows for personalized multi-user view through flexible layout of multiple windows, maximally utilizing the available screen real estate, personalized content synchronization through synchronization protocols and algorithms based on the operational transformation technique and a buffer compression algorithm based on the operational merging technique, maximally utilizing the available battery talk time and network resources, and personalized content retrieval through customizable search methods.

1. Introduction

Social software allows multiple users to be engaged in a common task or activities involving common interests or goals. Past years have witnessed the rapid growth of social software such as collaborative office productivity tools exemplified by *Google Docs* and *Codoxware*,[1] social media environments exemplified by *Wikepedia*, *Facebook*, and *Twitter*, and countless network-based multi-player games exemplified by *World of Warcraft* and *EverQuest*.

Triggered by *Apple*'s *iPhone*, mobile devices exemplified by smartphones have gained tremendous momentum, for example, IDG predicted that by the start of 2010 sales of smartphones within Australia would exceed 50% of new sales. However, compared to the widespread computer-based social software, the choices of mobile social software [5] are still limited. Most smartphone-based games are single-player games [12], such as *Angry Bird*. Social software available on smartphones are primarily email systems and non-real-time social media systems such as *Youtube*, *Facebook*, *Twitter*, and *Google+*.

Mobile real-time social software is virtually non-existent due to the device characteristics such as small screen real estate (e.g., ranging from 3 to 5 inches), limited battery talk time (e.g., typically up to 7 hours), scarce network resources (e.g., low bandwidth, high latency, and low stability), and inherent need for personalization [4], which present challenges to the design and implementation of effective and useful real-time mobile social software.

* Corresponding author.
E-mail addresses: haifeng.shen@flinders.edu.au (H. Shen), m.reilly@flinders.edu.au (M. Reilly).

[1] http://www.codoxware.com.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.10.008

First, it is non-trivial to design an effective multi-user interface on such a small screen, which allows each individual participant to focus on their own work and at the same time be given the flexibility of customizing the way they keep track of and interact with others. Second, it is non-trivial to devise an efficient and reliable solution to the synchronization of shared data across multiple devices given the scarce network resources and limited battery talk time. Last, The multi-user interface design and data synchronization solution also need to consider the device's memory capacity and limited multi-tasking capability.

In this article, we present a technical solution to these challenges using a smartphone-based real-time collaborative note-taking system – *GroupNotes* [17] – as an example. The solution allows for personalized multi-user view through flexible layout of multiple windows, maximally utilizing the available screen real estate, personalized content synchronization through synchronization protocols and algorithms based on the operational transformation technique [24] and a buffer compression algorithm based on the operational merging technique [19], maximally utilizing the available battery talk time and network resources, and personalized content retrieval through customizable search methods.

This solution decouples the multi-user interface from the underlying content synchronization. Users can customize the multi-user interface in such a way that best suites their working styles while remaining in a coherent collaborative session. They can customize the content synchronization policies to flexibly choose the way they wish to collaborate with others and the way others view their contributions to the shared task. They can also customize search methods to flexibly retrieve relevant content contributed by other users. The smartphone-based real-time collaborative note-taking system allows a small group of users to collaboratively take shared notes in real time. For example, a small group of students can use the system to participate in a real-time collaborative note-taking session in a lecture theatre using their own smartphones, and motivate, assist, and monitor each other in order to actively learn and keep everyone in the group engaged during the lecture, in a way that does not disrupt the lecture, either for the lecturer, or for other students [17].

The rest of the paper is organized as follows. The next section describes the personalized multi-window view interface. After that, we present the technical solution to personalized content synchronization followed by the operation buffer compression algorithm. We then briefly discuss personalized content retrieval through customizable search methods. Finally we conclude the paper with a summary of major contributions and future work.

2. Personalized multi-window view

We were informed by a preliminary needs finding survey that a user can only cognitively keep up with up to three other users in a real-time collaborative note-taking session [18]. In a session of up to four users, each participant is given the flexibility of customizing their own working style and the way they keep track of and interact with others. They may choose to work individually, cooperatively, or collaboratively in a session. To cater for that flexibility, we propose a multi-window solution to the multi-user interface design, where each participant's note area occupies a window, that is the number of windows in the multi-user interface is determined by the number of members who have currently joined in the real-time collaborative note-taking session.

The multi-window solution allows a participant to have a personalized multi-user view of the collaborative session on their own device, for example, Fig. 1 shows the default view of the user after they have finished making a note. The screen is divided into the editor area (upper part) and the radar view area (lower part). The device owner can view and make notes in the editor area, while editors from their group members who have joined the session are shown in the radar view area. In this figure, User 1's editor is shown in the editor area on the screen, while the editors from Users 2, 3 and 4, from left to right at the bottom of the screen, are shown in the radar view area. The background number in each editor, e.g., 1 in User 1's editor indicates that it is the first page of the notes.

A user chooses this view if their primary activity is to take their own notes, either individually rather than as part of a group, purely for the benefit of getting the digital form of the notes using their favorite device, or cooperatively where their notes contribute to the community notes of a group [28]. In this view, there is a single window in the editor area, giving the largest screen real estate possible for the device owner to edit or view the notes, while other users' windows are pushed to the lower part of the screen as miniatures, briefly informing the device owner of what others are doing in an unobtrusive way.

If the user wants to view more than one editor window simultaneously, they can drag one or more editors from the radar view area to the editor area, for example, Fig. 2 shows that the user has chosen to view two editor windows at the same time. This occurs by dragging User 2's miniature window (the first editor window in the radar view area of Fig. 1) up to the editor area. To return to a single editor windows, which may or may not be the device owner's editor, would require the reverse action, i.e. dragging the unwanted editor window down to the radar view area. Fig. 3 shows all four editor windows that take up all available screen real estate. It is worth pointing out that while a user can view up to four editor windows at the same time, when they start doing text entry in an editor, that editor window will occupy the entire editor area, pushing the rest of editor windows from the editor area to the radar view area, primarily due to the significant screen real estate taken by the soft keyboard.

A user may choose to view more than one editor window simultaneously if they want to work collaboratively with other participants in the session, where participants taking different roles in order to maximize their strength and cognitive power. For example, the first member is designated as the note-taker, who takes the entire notes, the second member is designated as the reviewer, who reviews and rectifies the notes, the third member is designated as the commentator, who

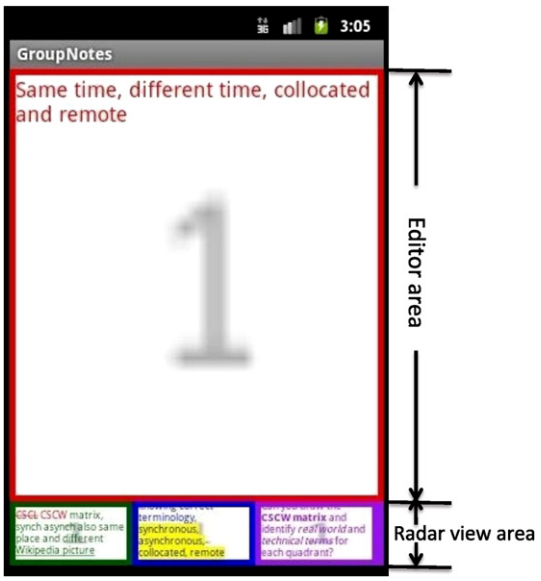


Fig. 1. 1 editor window and 3 miniature windows. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

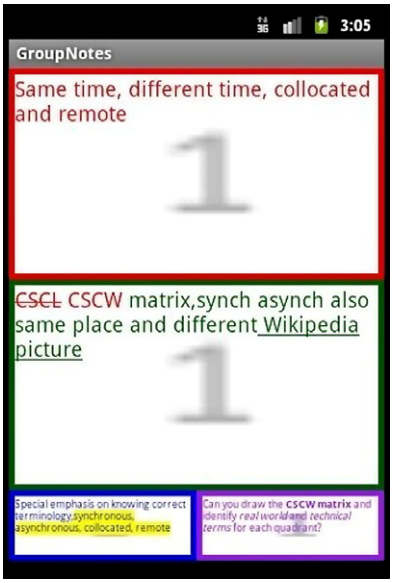


Fig. 2. 2 editor windows and 2 miniature windows. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

comments on the notes, and the fourth member is designated as the questioner, who develops questions about the notes. In this case, except the note-taker who is likely to choose a single editor window and focus on taking notes, all other participants – the reviewer, the commentator, and the questioner – are likely to choose to view multiple editor windows simultaneously.

It is worth clarifying that although each participant owns a dedicated editor window as their note area in a session, they can view and write to any note area at any time. To differentiate their editor windows and contributions to the same note area, participants are distinguished on the individual's device by the use of a unique color for each group member in the session. This color borders the editor window owned by that user and is illustrated in each of Figs. 1–3. The same color also identifies specific user generated content entered into another member's note area. For instance, in Fig. 2, User 1 (in red) performed a strikethrough on the text "CSCL" written by the owner of the second note area (in green) and "CSCW" is the new text written in this green note area, also in red. This allows other group members to determine who has made the change.

The proposed personalized multi-window view solution to the multi-user interface design is in a sharp contrast to the majority of computer-based real-time group editors such as *GROVE* [6], *REDUCE* [25], *JAMM* [1], and *WRACE* [21], where all

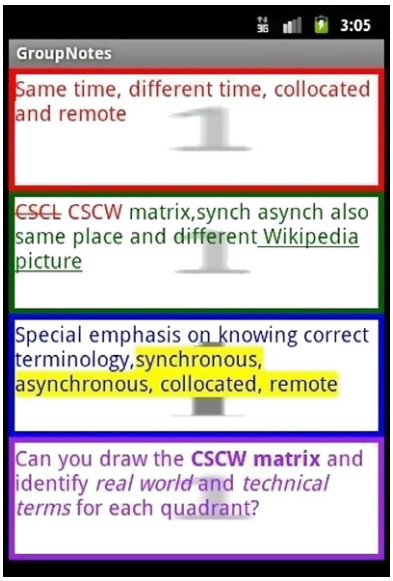


Fig. 3. All 4 editor windows. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

participants in a session share the same single editor window. Personalized view is supported through relaxed WYSIWIS (What You See is What I See), which allows participants to view different parts of the same editor window at the same time [7]. This solution works on large screens but is not suitable for handheld devices whose screens are rather small. Personalized view would not be supported if strict WYSIWIS were adopted, which forces all participants to view the same part of the editor window at any moment in time [23].

Some real-time group editors such as *MMM* [3] used multiple editor windows to represent a hierarchy of nested editors within each other. The number of editors did not correspond to the number of participants and all participants had the same view of all editors at any moment in time, that is strict WYSIWIS at all times. Furthermore, text edits into the same editor are floor controlled [8] and handled in the order in which they were received.

3. Personalized content synchronization

In a real-time collaborative note-taking session consisting of up to 4 members, each member owns a dedicated multi-page note. Each note can be jointly edited by the 4 users; therefore, the note-taking session is actually composed of 4 parallel collaborative editing sessions, one for each note. A synchronization solution is required to keep all members' notes consistent in the session.

Our content synchronization solution is based on the contextualization theory and extended from a data consistency maintenance solution for shared Web-based documents [21] to the synchronization of multiple notes in the same real-time collaborative note-taking session. Compared to other synchronization techniques, such as floor control [8], locking [10], transactions [2], causal ordering [16], and serialization [9], this solution can not only meet the three consistency properties required for collaborative editing systems: *convergency*, *causality preservation*, and *intention preservation* [25], but also the four requirements for satisfying users' diverse interaction and collaboration needs under complex and dynamic circumstances: *fast local response*, *total work preservation*, *unconstrained interaction*, and *customizable collaboration mode* [21].

In particular, a user can customize how the content should be synchronized by specifying three independent parameters.

1. *out*: whether they want to share their content with other members,
2. *in*: whether they want to accept the content shared by other members, and
3. *detail*: whether they want other members to replay their step-by-step updates or only the net effects on the shared content.

The *out* and *in* parameters will be discussed in the following synchronization protocols and algorithms based on the cornerstone technique called *operational transformation* [24]. The *detail* parameter determines whether a compression algorithm will be invoked to compress step-by-step updates into net effects. The compression algorithm will be discussed in the next section.

A central server will be used to synchronize the replicas of the shared content across all mobile devices, in addition to other functions such as repository management [22,29], session management [21,29], and note post-processing [18]. The owner of a device uses the note-taking app on their device to view or take notes. Updates on the notes – called operations – are broadcast to other devices for the synchronization of notes (subject to the setting of personalized content

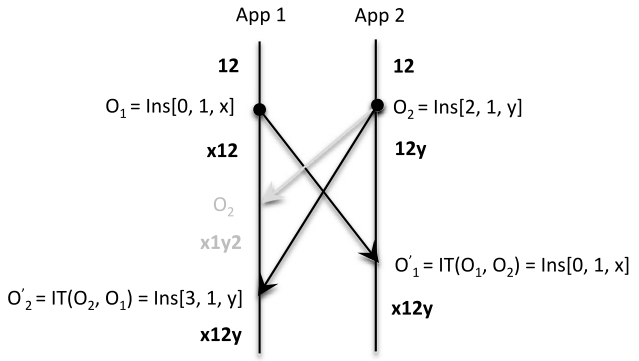


Fig. 4. Operational transformation.

synchronization policies). Before presenting the synchronization solution, we first introduce the operational transformation technique.

3.1. Operational transformation

There are two types of operations involved in editing a note: *Ins*/*Del*[*position*, *length*, *text*] denotes inserting/deleting a piece of *text* of *length* at the *position* in the note. Updating an attribute of a piece of text, e.g., highlighting the text, is represented by deletion of the text with the old attribute value followed by insertion of the same text with the new attribute value instead of by a new type of update operations, such as those used in collaborative word processors [27] because updating operations in note-taking are not as heavily used as in word processing.

As shown in Fig. 4, when operation $O_2 = Ins[2, 1, y]$ generated at *App* 2 arrives at *App* 1, it cannot be replayed as-is because the concurrent operation $O_1 = Ins[0, 1, x]$ has changed the context from that in which O_2 was defined, i.e., "12", to the new context "x12". Instead, O_2 needs to be transformed against O_1 in such a way that $O_2' = \mathbf{IT}(O_2, O_1) = Ins[3, 1, y]$ has effectively included the impact of the concurrent operation O_1.

$IT(O_a, O_b)$ is an *inclusion transformation* function that transforms operation O_a against operation O_b in such a way that the impact of O_b is effectively included in the parameters of the output operation O_a'. There are four instances of transformation functions, one for each pair of operation types, namely **IT_II**(O_a, O_b) (insert against insert), **IT_ID**(O_a, O_b) (insert against delete), **IT_DI**(O_a, O_b) (delete against insert), and **IT_DD**(O_a, O_b) (delete against delete).

We use **IT_ID** (O_a, O_b) as an example to illustrate operational transformation. More information about transformation functions can be found in these references [26,25]. For the good of presentation, for operation O = *Ins*/*Del*[*position*, *length*, *text*], $\mathbf{P}(O) = position$ denotes O's position parameter, $\mathbf{N}(O) = length$ denotes O's length parameter, $\mathbf{S}(O) = text$ denotes O's text parameter, and $\mathbf{T}(O) = Ins/Del$ denotes O's type parameter.

Algorithm 1 **IT_ID**(O_a, O_b): O_a'

```
1: if (P(O_a) ⩽ P(O_b) or N(O_b) == 0) then
2:     return
3: else if (P(O_a) > P(O_b) + N(O_b)) then
4:     P(O_a) ← P(O_a) − N(O_b)
5: else if (P(O_a) == P(O_b) + N(O_b)) then
6:     P(O_a) ← P(O_b)
7: else
8:     O_a ← I {I is an identity (null) operation}
9: end if
```

3.2. Operation broadcast

The synchronization solution consists of an operation broadcast protocol, an operation replay algorithm, and a set of session management protocols. A note synchronization process consists of two sub-processes: broadcast local operations and replay remote operations. As shown in Fig. 5, the server runs m ($m \geqslant 1$) collaborative note-taking sessions; each session has up to 4 collaboratively edited notes. There are n ($n \geqslant 1$) apps connected to the server and up to 4 apps, e.g. App i, j, k, and l ($1 \leqslant i, j, k, l \leqslant n$), can share the same session, e.g. session p ($1 \leqslant p \leqslant m$).

For each note, the server maintains a master note *MN*, a master incoming operation buffer *MIB* storing all operations that should be executed on *MN* to get its latest state, and a server-side incoming operation buffer for each app involved in the same note, e.g., SIB_i, SIB_j, SIB_k, and SIB_l in session p, storing remote operations that have been received by the server but are yet to be received by the corresponding app. Each app can write into any of the 4 notes at any time, therefore it needs to separately synchronize these 4 notes. For each note, the app maintains a replica of the note *RN*, an outgoing

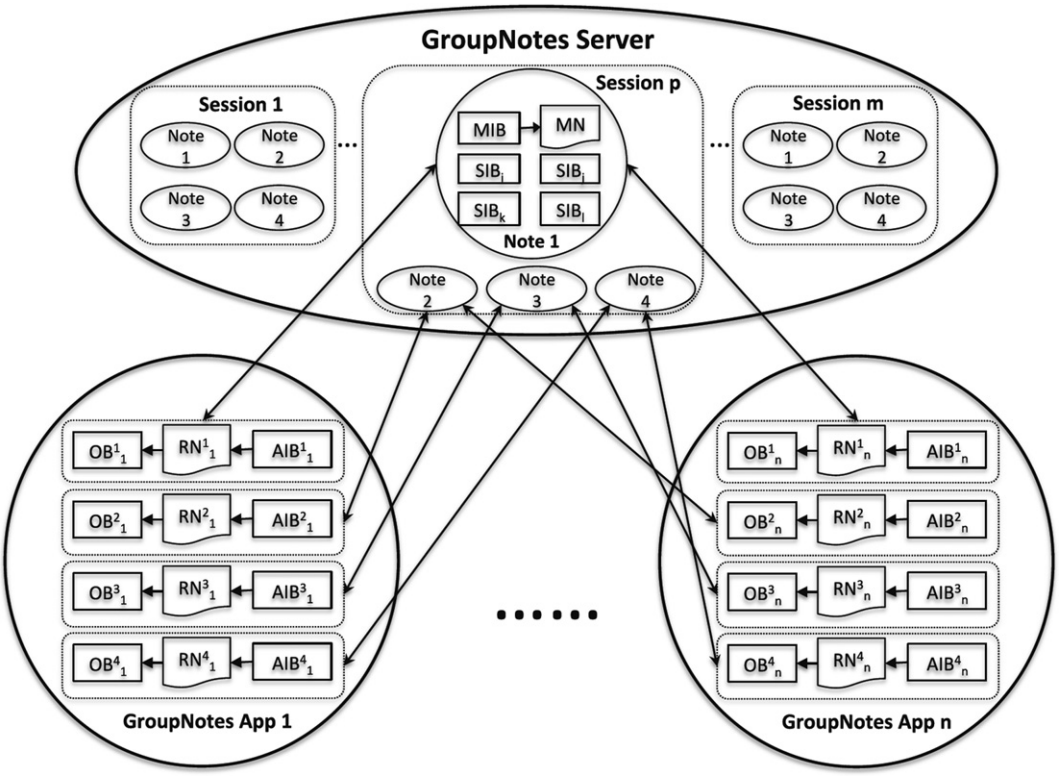


Fig. 5. Note synchronization.

operation buffer *OB* storing locally generated operations on *RN*, and an app-side incoming operation buffer *AIB* storing remote operations that have already been received by the app and will be replayed on *RN*. In Fig. 5, RN^i_j, OB^i_j, and AIB^i_j ($1 \leqslant i \leqslant 4$ and $1 \leqslant j \leqslant n$) are App j's replica, *OB*, and *AIB* for note i respectively.

Suppose App k ($1 \leqslant k \leqslant n$) is involved in note r ($1 \leqslant r \leqslant 4$) in the collaborative note-taking session p ($1 \leqslant p \leqslant m$), the following protocol is executed by the app to broadcast local operations generated on note replica RN^r_k from OB^r_k. This protocol broadcasts a sequence of operations generated by each app instead of one at a time, significantly reducing the consumption of the mobile device talk time and network resources. Separation of outgoing and incoming operation buffers and separation of app-side and server-side buffers, reinforced by the note replicas in the server and across the apps, make the protocol resilient to the network disruptions in the mobile devices.

Protocol 1. Operation broadcast protocol

1. At App k, move all available operations in OB^r_k to the operation sequence $\overrightarrow{OB}^r_k$. If AIB^r_k is not empty, invoke the procedure **SLOT**($\overrightarrow{OB}^r_k$, AIB^r_k) to transform the operation sequence $\overrightarrow{OB}^r_k$ with the operation sequence in AIB^r_k.
2. App k establishes a network connection with the server and then sends a request $\langle BROADCAST, k, p, r, \overrightarrow{OB}^r_k\rangle$ to the server.
3. When the server receives the request from App k, it performs an atomic synchronization process on note r in session p:
   (a) lock note r, including *MN*, *MIB* and all *SIB*'s, e.g., SIB_i, SIB_j, SIB_k, and SIB_l ($1 \leqslant i, j, k, l \leqslant n$);
   (b) if SIB_k is not empty, invoke **SLOT**($\overrightarrow{OB}^r_k$, SIB_k) to transform $\overrightarrow{OB}^r_k$ with the operation sequence in SIB_k;
   (c) append $\overrightarrow{OB}^r_k$ to *MIB*, SIB_i, SIB_j, and SIB_l;
   (d) if SIB_k is not empty, move all operations in SIB_k to the operation sequence $\overleftarrow{SIB}_k$ and send a response $\langle BROADCAST, \overleftarrow{SIB}_k\rangle$ to App k, otherwise, send a dummy response $\langle BROADCAST\rangle$ to App k; and
   (e) unlock note r in session p.
4. When App k receives the response from the server, if the response is not dummy, it appends the piggyback operation sequence $\overleftarrow{SIB}_k$ to AIB^r_k.

The **SLOT** transformation control algorithm [21,20], which symmetrically transforms two context-equivalent sequences $Sq_a = [O_{a,0} \cdots O_{a,m-1}]$ ($m = |Sq_a|$) and $Sq_b = [O_{b,0} \cdots O_{a,n-1}]$ ($n = |Sq_b|$), and returns transformed ones Sq^b_a and Sq^a_b, is defined in Algorithm 2. The **SIT** transformation function, which symmetrically transforms two context-equivalent operations $O^j_{a,i}$ and $O^i_{b,j}$ ($0 \leqslant i < m$, $0 \leqslant j < n$), and returns transformed ones $O^{j+1}_{a,i}$ and $O^{i+1}_{b,j}$, is defined Algorithm 3.

Algorithm 2 SLOT(Sq_a, Sq_b): (Sq_a^b, Sq_b^a)

$Sq_a^b \leftarrow Sq_a$
$Sq_b^a \leftarrow Sq_b$
for ($i \leftarrow 0$; $i < |Sq_a^b|$; i++) **do**
  for ($j \leftarrow 0$; $j < |Sq_b^a|$; j++) **do**
    $O_{a,i}^{j} \leftarrow Sq_a^b[i]$
    $O_{b,j}^{i} \leftarrow Sq_b^a[j]$
    $(O_{a,i}^{j+1}, O_{b,j}^{i+1}) \leftarrow$ **SIT**$(O_{a,i}^{j}, O_{b,j}^{i})$
    $Sq_a^b[i] \leftarrow O_{a,i}^{j+1}$
    $Sq_b^a[j] \leftarrow O_{b,j}^{i+1}$
  end for
end for
return (Sq_a^b, Sq_b^a)

Algorithm 3 SIT($O_{a,i}^{j}$, $O_{b,j}^{i}$): ($O_{a,i}^{j+1}$, $O_{b,j}^{i+1}$)

$O_{a,i}^{j+1} \leftarrow$ **IT**$(O_{a,i}^{j}, O_{b,j}^{i})$
$O_{b,j}^{i+1} \leftarrow$ **IT**$(O_{b,j}^{i}, O_{a,i}^{j})$
return $(O_{a,i}^{j+1}, O_{b,j}^{i+1})$

3.3. Operation replay

The other sub-process involved in a note synchronization is to replay remote operations stored in an app's *AIB*. The following operation replay algorithm executes remote operations in AIB_k^r to complete the synchronization process on note r by app k.

Algorithm 4 Operation replay algorithm

1: **if** ($|AIB_k^r| == 0$) **then**
2: **return**
3: **end if**
4: $start \leftarrow 0$
5: **if** ($|OB_k^r| > start$) **then**
6: $end \leftarrow |OB_k^r| - 1$
7: **SLOT**$(AIB_k^r, OB_k^r[start, end])$
8: $start \leftarrow end + 1$
9: **end if**
10: **repeat**
11: *give_way*()
12: *lock_replica*()
13: **if** ($|OB_k^r| > start$) **then**
14: $end \leftarrow |OB_k^r| - 1$
15: **SLOT**$(AIB_k^r, OB_k^r[start, end])$
16: $start \leftarrow end + 1$
17: **end if**
18: $O \leftarrow AIB_k^r[0]$
19: *execute*(O)
20: AIB_k^r.*remove*(0)
21: *unlock_replica*()
22: **until** ($|AIB_k^r| == 0$ **or** *time to broadcast*)

It is worth pointing out that because local and remote operations need to modify the same replica of note r at App k, execution of local operations and replay of remote operations must be mutually exclusive. In case of contention between local and remote operations, local operations must be given the priority to ensure good local response. Otherwise, if all remote operations in AIB_k^r were replayed as a continual stream, local operations would suffer starvation, resulting in poor local response. To minimize the impact on the local response, each remote operation in AIB_k^r should "give way" to new local operations before being replayed.

3.4. Personalized synchronization policies

If both *out* and *in* are on, each synchronization process initiated by the app will first execute the operation broadcast protocol to broadcast local operations from *OB* and then invoke the operation replay algorithm to replay remote operations

from *AIB* on the local replica of the note. If *out* is on but *in* is off, each synchronization process will only execute the operation broadcast protocol to broadcast local operations from *OB* but will not invoke the operation replay algorithm to replay remote operations. Nonetheless, remote operations are still kept in *AIB* so that they will be transformed with for achieving contextualization when the operation broadcast protocol is executed. If *out* is off but *in* is on, each synchronization process will first execute the operation broadcast protocol but with a dummy request (i.e., the request does not piggyback the operation sequence in *OB*) and then invoke the operation replay algorithm to replay remote operations from *AIB*. Nonetheless, local operations are still kept in *OB* so that they will be transformed with for achieving contextualization when the operation replay algorithm is executed.

Two points are worthing clarifying. First, consistency is still achieved even though each app may end up with a different replica of the same note because causality preservation (ensured by orderly broadcast and transformation) and intention preservation (ensured by transformation) are not affected, and if all operations were executed on all replicas, they would become identical (same as the master copy). Second, if a user wants to save a note, after producing the latest master note by incorporating all broadcast operations on the server, the user will be prompted to save their local replica into their personal space on the server or their mobile device itself as it is likely to be different from the master note on the server.

3.5. Session management protocols

The session creation protocol is to be executed when a user wants to create a new collaborative note-taking session, the session joining protocol is to be executed when a new user (i.e. a latecomer) wants to join an ongoing session, the session saving protocol is to be executed when a user wants to produce the latest master copies of all the shared notes for the session, and the session leaving protocol is to be executed when a user wants to quit from an ongoing session.

Protocol 2. Session creation: a new app creates a new session

1. The app establishes a network connection with the server and sends a request $\langle CREATE \rangle$ to the server.
2. When the server receives the request from the app, it performs the following session creation process:
   (a) assign a session id *sid* ($1 \leqslant sid \leqslant$ m), a note id *nid* ($1 \leqslant nid \leqslant 4$) within the session, and an app id *aid_nid* ($1 \leqslant aid\_nid \leqslant$ n) for the app that owns note *nid*;
   (b) create session *sid* that contains note *nid*, including the master copy of the note *MN*, the master incoming buffer *MIB*, and the sever-side incoming buffer SIB_{aid_nid} for app *aid_nid*; and
   (c) send a response $\langle CREATE, sid, nid, aid\_nid, MN \rangle$ to the app.
3. When the app receives the response from the server, it first tags itself with the assigned *sid*, *nid*, and *aid_nid*, then creates $OB^{nid}_{aid\_nid}$ and $AIB^{nid}_{aid\_nid}$, and finally opens $RN^{nid}_{aid\_nid} = MN$.

Protocol 3. Session joining: a new app joins session p

1. The app establishes a network connection with the server and sends a request $\langle JOIN, p \rangle$ ($1 \leqslant p \leqslant m$) to the server.
2. When the server receives the request from the app, it performs the following session joining process on session p:
   (a) lock session p, including all the existing notes, e.g., note i owned by app *aid_i* and note j owned by app *aid_j* ($1 \leqslant i \neq j \leqslant 4$ and $1 \leqslant aid\_i \neq aid\_j \leqslant n$);
   (b) assign a note id *nid* ($1 \leqslant nid \leqslant 4$ and $nid \neq i \neq j$) and an app id *aid_nid* ($1 \leqslant aid\_nid \leqslant n$ and $aid\_nid \neq aid\_i \neq aid\_j$);
   (c) create note *nid*, including *MN*, *MIB*, and SIB_{aid_i}, SIB_{aid_j}, and SIB_{aid_nid};
   (d) within notes i and j, create SIB_{aid_nid} and then execute and remove all operations in *MIB* to produce the latest *MN*;
   (e) send a response $\langle JOIN, nid, aid\_nid, MN, i, note\ i\text{'s } MN, j, note\ j\text{'s } MN \rangle$ to the app and a response $\langle JOIN, nid, aid\_nid, MN \rangle$ to apps *aid_i* and *aid_j* and; and
   (f) unlock session p.
3. When the app receives the response from the server, it first tags itself with p, *nid*, and *aid_nid*, then creates $OB^{i}_{aid\_i}$ and $AIB^{i}_{aid\_i}$, $OB^{j}_{aid\_j}$ and $AIB^{j}_{aid\_j}$, $OB^{nid}_{aid\_nid}$ and $AIB^{nid}_{aid\_nid}$, and finally opens $RN^{nid}_{aid\_nid} = note\ nid\text{'s } MN$, $RN^{i}_{aid\_nid} = note\ i\text{'s } MN$, and $RN^{j}_{aid\_nid} = note\ j\text{'s } MN$.
4. When app *aid_i* (or *aid_j*) receives the response from the server, it creates $OB^{nid}_{aid\_i}$ (or $OB^{nid}_{aid\_j}$) and $AIB^{nid}_{aid\_i}$ (or $AIB^{nid}_{aid\_j}$), and then opens $RN^{nid}_{aid\_i}$ (or $RN^{nid}_{aid\_j}$) $= note\ nid\text{'s } MN$.

Protocol 4. Session saving: app *aid* **saves session** p

1. App *aid* executes the operation broadcast protocol for each non-empty *OB*.
2. App *aid* establishes a network connection with the server and sends a request $\langle SAVE, p \rangle$ to the server.
3. When the server receives the request from app *aid*, it performs the following session saving process on session p:
   (a) lock session p, including all the existing notes;
   (b) execute and remove all operations in each note's *MIB* to produce the latest master copy *MN*;

(c) send a dummy response ⟨*SAVE*⟩ to the app; and
(d) unlock session p.
4. When the app receives the response from the server, it takes no action.

Protocol 5. Session leaving: app *aid* leaves session p

1. App *aid* locks the local replicas of all the shared notes.
2. App *aid* executes the operation broadcast protocol for each non-empty *OB*.
3. App *aid* establishes a network connection with the server and sends a request ⟨*LEAVE*, *aid*, p⟩ to the server.
4. When the server receives the request from the app, it performs the following session leaving process on session p:
   (a) lock session p, including all the existing notes;
   (b) execute all operations in the *MIB* from note *nid* owned by app *aid* to produce the latest master copy *MN*;
   (c) back up *MN* on the server and then remove app *aid*'s note *nid* from session p;
   (d) remove app *aid* from other notes in session p, including SIB_{aid} in each of these notes;
   (e) send a dummy response ⟨*LEAVE*⟩ to the app and a response ⟨*LEAVE*, *aid*, *nid*⟩ to other apps in session p; and
   (f) unlock session p.
5. When the app receives the response from the server, it quits. When other apps, e.g., App k, receive the response from the server, they remove OB_k^{nid} and AIB_k^{nid} and RN_k^{nid} from the interface.

4. Operation buffer compression

The network connection between an app and the server is based on Wi–Fi, where the network resources are always limited. Frequent broadcast and replay of every individual operation should be avoided unless it is really needed as it will consume too much network bandwidth as well as the precious battery talk time of the device running the app. It may also be unnecessary if users are only interested in what note their peers are taking rather than micro-step operations. Therefore, our personalized content synchronization policy allows a user to choose from broadcasting every individual operation made on a note or just broadcasting the net effects of these operations through the *detail* parameter. The technical solution behind this policy is an operation buffer compression algorithm that can compress operations in each of an app's *OB*'s to be operations of net effects. That is, a user can set the *detail* parameter for each of the notes in their app. If a note's *detail* parameter is off, the corresponding *OB* in this app will be compressed before being broadcast to other apps.

The buffer compression issue is similar to the log compression issue in operation-based source code control systems [13] and distributed file systems [15]. For example, for replicas to effectively manage the storage resources of their write-logs in an anti-entropy protocol for the propagation of write operations between weakly consistent storage replicas [15], each replica can independently decide when and how aggressively to prune a prefix of its write-log subject to the constraint that only stable writes can get discarded. An important consequence of this approach is that a replica may discard write operations that have not been propagated to other replicas, leading to inconsistency and the solution is to transfer the full database state from one replica to the other if the two replicas are far from consistent. As the solution sometimes needs to transfer a full, large document over the network, it is not suitable for the mobile devices, where network resources are limited. Furthermore, it cannot satisfy the *total work preservation* requirement in a collaborative note-taking session.

Lippe and Oosterom proposed the concept of redundant operations in object-oriented operation-based merging [11]. Although the objective of removing redundant operations is to remove unnecessary conflicts and to speed up conflict detection, the concept of redundant operations has inspired us to devise the operational merging technique for compressing logs in text-oriented operation-based source code control systems [19]. The proposed compression algorithm is extended from this operational merging technique to support compressing operation buffers in mobile devices.

It is worth clarifying that the compression algorithm is based on operation relationships in the buffer and their context in relation to the shared note. A compressed buffer can be further compressed using a standard lossless compression algorithm such as LZ77 [30] to reduce the size of the buffer in order to reduce the consumption of network resources, which however cannot be used to produce operations of net effects alone.

4.1. Operation relationships

Definition 1 *(Operation context).* Given an operation O, its context, denoted by Υ_O, is the state of the note on which O is defined.

Definition 2 *(Operation context equivalent relation).* Given two operations O_a and O_b, O_a is context-equivalent to O_b, denoted by $O_a \sqcup O_b$, iff $\Upsilon_{O_a} = \Upsilon_{O_b}$.

Definition 3 *(Operation context preceding relation).* Given two operations O_a and O_b, O_a is context-preceding O_b, or O_b is context-succeeding O_a, denoted by $O_a \mapsto O_b$, iff $\Upsilon_{O_b} = \Upsilon_{O_a} \vdash O_a$, where '$\vdash$' is the operation execution operator.

Υ_{O_a} is the context (i.e., state of the note) on which O_a is defined. After the execution of O_a on Υ_{O_a}, the new context can be described by $\Upsilon_{O_a} \vdash O_a$. If O_b is defined on this new context, i.e., $\Upsilon_{O_b} = \Upsilon_{O_a} \vdash O_a$, then O_b is context-succeeding O_a. In general, if the initial context of a note is Φ and n operations $O_1, \ldots, O_n$ have been executed in sequence, i.e., $OB = [O_1, \ldots, O_n]$, then the final context can be described by $\Phi \vdash O_1 \vdash \cdots \vdash O_n$. For $\forall i \in \{2, \ldots, n\}$, $O_{i-1} \mapsto O_i$ because $\Upsilon_{O_1} = \Phi$ and $\Upsilon_{O_i} = \Upsilon_{O_{i-1}} \vdash O_{i-1}$.

The following definitions are used to describe the relationships between any two operations in an *OB*.

Definition 4. Operation overlapping relation "$\oplus$".

Given two operations O_a and O_b where $O_a \mapsto O_b$, O_a and O_b are *overlapping*, denoted as $O_a \oplus O_b$, **iff** (if and only if) one of following conditions holds:

1. $\mathbf{T}(O_a) = \mathbf{T}(O_b) = \mathit{Ins}$, and $\mathbf{P}(O_a) < \mathbf{P}(O_b) < \mathbf{P}(O_a) + \mathbf{N}(O_a)$.
2. $\mathbf{T}(O_a) = \mathbf{T}(O_b) = \mathit{Del}$, and $\mathbf{P}(O_b) < \mathbf{P}(O_a) < \mathbf{P}(O_b) + \mathbf{N}(O_b)$.
3. $\mathbf{T}(O_a) = \mathit{Ins}$ and $\mathbf{T}(O_b) = \mathit{Del}$, and $\mathbf{P}(O_a) \leqslant \mathbf{P}(O_b) < \mathbf{P}(O_a) + \mathbf{N}(O_a)$ or $\mathbf{P}(O_b) \leqslant \mathbf{P}(O_a) < \mathbf{P}(O_b) + \mathbf{N}(O_b)$.

4.2. Operational merging

Two operations O_a and O_b are overlapping if their effect regions are overlapping. First, if an insertion operation O_b inserts text that falls into the effect region of a previous insertion operation O_a, then O_a and O_b are overlapping. Second, if a deletion operation O_a deletes a range that falls into the effect region of a later deletion operation O_b, then O_a and O_b are overlapping. Third, if an insertion operation O_a inserts text which or part of which falls into the effect region of a later deletion operation O_b, then O_a and O_b are overlapping. Finally, under no circumstance could an insertion operation overlap with a previous deletion operation because there is no way for text to be inserted into non-existent text (i.e., text that has already been deleted).

Definition 5. Operation adjacent relation "$\ominus$".

Given two operations O_a and O_b where $O_a \mapsto O_b$, O_a and O_b are *adjacent*, denoted as $O_a \ominus O_b$, **iff** one of the following conditions holds:

1. $\mathbf{T}(O_a) = \mathbf{T}(O_b) = \mathit{Ins}$, and $\mathbf{P}(O_b) = \mathbf{P}(O_a)$ or $\mathbf{P}(O_b) = \mathbf{P}(O_a) + \mathbf{N}(O_a)$.
2. $\mathbf{T}(O_a) = \mathbf{T}(O_b) = \mathit{Del}$, and $\mathbf{P}(O_a) = \mathbf{P}(O_b) + \mathbf{N}(O_b)$ or $\mathbf{P}(O_a) = \mathbf{P}(O_b)$.

The same type of two operations are adjacent if their effect regions are adjacent. If an insertion operation O_b inserts a string that is adjacent to the string inserted by a previous insertion operation O_a, then O_a and O_b are adjacent. If a deletion operation O_b deletes a range that is adjacent to the range deleted by a previous deletion operation O_a, then O_a and O_b are adjacent.

Definition 6. Operation disjointed relation "$\odot$".

Given two operations O_a and O_b, O_a and O_b are *disjointed*, denoted as $O_a \odot O_b$, **iff** neither $O_a \oplus O_b$ nor $O_a \ominus O_b$.

Two adjacent operations can be merged into one operation by concatenating their effect regions. In this way, the number of operations in an *OB* can be reduced by one. The same type of two overlapping operations can be merged into one operation by combining their effect regions. In this way, the number of operations in the *OB* can be reduced by one. Different types of two overlapping operations can be merged in such a way that the overlapping region is removed from both operations. In this way, the size of the *OB* can be reduced and the number of operations in the log could be reduced by one or two if the effect region of one operation totally falls into the effect region of the other operation or the effect regions of the two operations are completely overlapping. The following functions are defined to merge two operations in an *OB*.

OM_II(O_a, O_b) is defined to merge two insertion operations O_a and O_b where $O_a \mapsto O_b$. If $O_a \oplus O_b$ or $O_a \ominus O_b$, O_a and O_b will be merged into a single insertion operation O'_a that integrates the effect regions covered by both O_a and O_b. **OM_DD**(O_a, O_b) is defined to merge two deletion operations O_a and O_b where $O_a \mapsto O_b$. If $O_a \oplus O_b$ or $O_a \ominus O_b$, O_a and O_b will be merged into a single deletion operation O'_a that integrates the effect regions covered by both O_a and O_b. **OM_ID**(O_a, O_b) function is defined to merge an insertion operation O_a and a deletion operation O_b where $O_a \mapsto O_b$. If $O_a \oplus O_b$, the overlapping region will be removed from both O_a and O_b in such a way that the common substring inserted by O_a but later deleted by O_b is eliminated from both O_a and O_b. **OM_ID**(O_a, O_b) is the most effective merging function that can dramatically reduce the size and the number of operations in an *OB* by removing redundant operations or redundant information in operations.

Algorithm 5 OM(O_a, O_b): (O'_a, O'_b)

Require: $O_a \mapsto O_b$
Ensure: $O'_a \odot O'_b$
  if ($\mathbf{T}(O_a) == Ins$ **and** $\mathbf{T}(O_b) == Ins$) **then**
    return $OM\_II(O_a, O_b)$
  else if ($\mathbf{T}(O_a) == Ins$ **and** $\mathbf{T}(O_b) == Del$) **then**
    return $OM\_ID(O_a, O_b)$
  else if ($\mathbf{T}(O_a) == Del$ **and** $\mathbf{T}(O_b) == Del$) **then**
    return $OM\_DD(O_a, O_b)$
  else
    return (O_a, O_b)
  end if

Algorithm 6 OM_DD(O_a, O_b): (O'_a, O'_b)

  if ($\mathbf{P}(O_a) \geqslant \mathbf{P}(O_b)$ **and** $\mathbf{P}(O_a) \leqslant \mathbf{P}(O_b) + \mathbf{N}(O_b)$) **then**
    $head \leftarrow$ **substring**($\mathbf{S}(O_b)$, 0, $\mathbf{P}(O_a) - \mathbf{P}(O_b)$)
    $tail \leftarrow$ **substring**($\mathbf{S}(O_b)$, $\mathbf{P}(O_a) - \mathbf{P}(O_b)$, $\mathbf{N}(O_b)$)
    $\mathbf{T}(O'_a) \leftarrow Del$; $\mathbf{P}(O'_a) \leftarrow \mathbf{P}(O_b)$
    $\mathbf{N}(O'_a) \leftarrow \mathbf{N}(O_a) + \mathbf{N}(O_b)$
    $\mathbf{S}(O'_a) \leftarrow head + \mathbf{S}(O_a) + tail$
    return $(O'_a, \mathbf{I})$
  else
    return (O_a, O_b)
  end if

Algorithm 7 OM_II(O_a, O_b): (O'_a, O'_b)

  if ($\mathbf{P}(O_b) \geqslant \mathbf{P}(O_a)$ **and** $\mathbf{P}(O_b) \leqslant \mathbf{P}(O_a) + \mathbf{N}(O_a)$) **then**
    $head \leftarrow$ **substring**($\mathbf{S}(O_a)$, 0, $\mathbf{P}(O_b) - \mathbf{P}(O_a)$)
    $tail \leftarrow$ **substring**($\mathbf{S}(O_a)$, $\mathbf{P}(O_b) - \mathbf{P}(O_a)$, $\mathbf{N}(O_a)$)
    $\mathbf{T}(O'_a) \leftarrow Ins$; $\mathbf{P}(O'_a) \leftarrow \mathbf{P}(O_a)$
    $\mathbf{N}(O'_a) \leftarrow \mathbf{N}(O_a) + \mathbf{N}(O_b)$
    $\mathbf{S}(O'_a) \leftarrow head + \mathbf{S}(O_b) + tail$
    return $(O'_a, \mathbf{I})$
  else
    return (O_a, O_b)
  end if

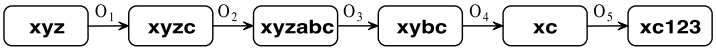


Fig. 6. A buffer with 5 editing operations.

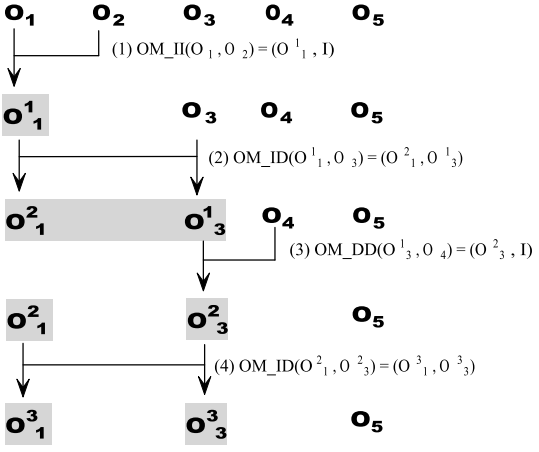


Fig. 7. Buffer compression through operational merging.

For example, as shown in Fig. 6, the note initially contained a string *xyz* and was transformed to another string *xc123* by a list of user-issued editing operations stored in $OB = [O_1, O_2, O_3, O_4, O_5]$. By applying operational merging to the neighboring operations in $OB = [O_1, O_2, O_3, O_4, O_5]$, OB can be compressed step by step as shown in Fig. 7.

1. $O_1 \ominus O_2$, which can be merged by **OM_II**$(O_1, O_2) = (O_1^1, I)$ where $O_1^1 =$ **Ins**$[3, 3, abc]$. After that, $OB = [O_1^1, O_3, O_4, O_5]$.
2. $O_1^1 \oplus O_3$, which can be merged by **OM_ID**$(O_1^1, O_3) = (O_1^2, O_3^1)$ where $O_1^2 =$ **Ins**$[3, 2, bc]$ and $O_3^1 =$ **Del**$[2, 1, z]$. After that, $OB = [O_1^2, O_3^1, O_4, O_5]$.
3. $O_3^1 \oplus O_4$, which can be merged by **OM_DD**$(O_3^1, O_4) = (O_3^2, I)$ where $O_3^2 =$ **Del**$[1, 3, yzb]$. After that, $OB = [O_1^2, O_3^2, O_5]$.
4. $O_1^2 \oplus O_3^2$, which can be merged by **OM_ID**$(O_1^2, O_3^2) = (O_1^3, O_3^3)$ where $O_1^3 =$ **Ins**$[3, 1, c]$ and $O_3^3 =$ **Del**$[1, 2, yz]$. After that, $OB = [O_1^3, O_3^3, O_5]$.

Because $O_1^3 \odot O_3^3$ and $O_3^3 \odot O_5$, OB could not be further compressed by applying operational merging on neighboring operations. The number of operations in OB has been reduced from 5 to 3, achieving 40% reduction. The size of the buffer is the total sizes of individual operations measured in bytes. For operation $O =$ **Ins/Del**$[P, N, S]$, one byte is used to describe O's type parameter $\mathbf{T}(O)$, four bytes are used to describe O's position parameter $\mathbf{P}(O)$, four bytes are used to describe O's length parameter $\mathbf{N}(O)$, and each character is allocated with one byte for O's text parameter $\mathbf{S}(O)$. So for $OB = [O_1, O_2, O_3, O_4, O_5]$, its size is 65 bytes. After OB has been compressed to $[O_1^3, O_3^3, O_5]$, its size is 33 bytes, achieving 49% reduction. The question is whether there is any room for OB to be further compressed by applying operational merging.

Algorithm 8 **OM_ID**(O_a, O_b): (O_a', O_b')

```
if (P(O_b) ⩽ P(O_a) and P(O_b) + N(O_b) ⩾ P(O_a) + N(O_a)) then
  if (P(O_b) == P(O_a) and P(O_b) + N(O_b) == P(O_a) + N(O_a)) then
    return (I, I)
  else
    head ← substring(S(O_b), 0, P(O_a) − P(O_b))
    tail ← substring(S(O_b), P(O_a) + N(O_a) − P(O_b), N(O_b))
    T(O_b') ← Del; P(O_b') = P(O_b)
    N(O_b') ← N(O_b) − N(O_a); S(O_b') ← head + tail
    return (I, O_b')
  end if
else if (P(O_b) ⩾ P(O_a) and P(O_b) + N(O_b) ⩽ P(O_a) + N(O_a)) then
  head ← substring(S(O_a), 0, P(O_b) − P(O_a))
  tail ← substring(S(O_a), P(O_b) + N(O_b) − P(O_a), N(O_a))
  T(O_a') ← Ins; P(O_a') ← P(O_a)
  N(O_a') ← N(O_a) − N(O_b); S(O_a') ← head + tail
  return (O_a', I)
else if (P(O_b) > P(O_a) and P(O_b) + N(O_b) > P(O_a) + N(O_a)) then
  T(O_a') ← Ins; P(O_a') = P(O_a)
  N(O_a') ← P(O_b) − P(O_a)
  S(O_a') ← substring(S(O_a), 0, P(O_b) − P(O_a))
  T(O_b') ← Del; P(O_b') = P(O_b)
  N(O_a') ← P(O_b) + N(O_b) − P(O_a) − N(O_a)
  S(O_a') ← substring(S(O_b), P(O_a) + N(O_a) − P(O_b), N(O_b))
  return (O_a', O_b')
else if (P(O_b) < P(O_a) and P(O_b) + N(O_b) < P(O_a) + N(O_a)) then
  T(O_a') ← Ins; P(O_a') = P(O_a)
  N(O_a') ← P(O_a) + N(O_a) − P(O_b) − N(O_b)
  S(O_a') ← substring(S(O_a), P(O_b) + N(O_b) − P(O_a), N(O_a))
  T(O_b') ← Del; P(O_b') = P(O_b)
  N(O_a') ← P(O_a) − P(O_b)
  S(O_a') ← substring(S(O_b), 0, P(O_a) − P(O_b))
  return (O_a', O_b')
else
  return (O_a, O_b)
end if
```

4.3. The compression algorithm

Definition 7. Maximally compressed OB "Ω_{OB}".

Given an OB, Ω_{OB} denotes its *maximally compressed* form in which for $\forall O_i, O_j \in OB$ $(0 \leqslant i, j < |OB|$ and $i \neq j)$, $O_i \odot O_j$.

We devised a compression algorithm based on operational merging, which can achieve maximal compression. First, the algorithm exhaustively examines every two operations in an OB, including those that are not physically located one after another in the buffer. For the example in Fig. 6, the compressed OB is $[O_1^3, O_3^3, O_5]$, where $O_1^3 \odot O_3^3$ and $O_3^3 \odot O_5$, but the relationship between O_1^3 and O_5 is not examined. We adopt the operational transformation technique to reshuffle an OB through the **LTranspose** (OB, i, j) $(0 \leqslant i < j < |OB|)$ function, which contextually swaps operations $OB[i]$ and $OB[j]$.

Applying **LTranspose** $(OB, 1, 2)$ to the aforesaid example would result in a reshuffled $OB = [O_3^4, O_1^4, O_5]$, where $O_3^4 = Del[1, 2, yz]$, $O_1^4 = Ins[1, 1, c]$, and $O_3^4 \mapsto O_1^4 \mapsto O_5$. It is clear that $O_1^4 \ominus O_5$ and the OB can be further compressed to $OB =$

Algorithm 9 LTranspose(OB, i, j)

```
for (k ← j; k > i; k ← k − 1) do
    O_{k−1} ← OB[k − 1]; O_k ← OB[k]
    Transpose(O_{k−1}, O_k)
    OB[k − 1] ← O_k
    OB[k] ← O_{k−1}
end for
```

Algorithm 10 Transpose(O_a, O_b): (O'_b, O'_a)

Require: $O_a \mapsto O_b$
Ensure: $O'_b \mapsto O'_a$

```
O'_b ← IT (O_b, Ō_a) {Ō_a is O_a's inverse}
O'_a ← IT (O_a, O'_b)

return (O'_b, O'_a)
```

Algorithm 11 COMET(OB): ($\widetilde{OB}$)

```
OB̃^d ← COMEType(OB, Del)
OB̃^i ← COMEType(OB, Ins)

return OB̃ ← OB̃^d + OB̃^i
```

Algorithm 12 COMEType(OB, Type): ($\widetilde{OB}^x$)

```
while ((i ← lastOp(OB, Type)) ⩾ 0 and merged == false) do
    if (i == 0) then
        OB̃^x.add(OB[i])
        OB.remove(i)
        merged ← true
    else
        for (j ← i − 1; j > 0; j ← j − 1) do
            (O_i, O_j) ← OM(OB[j], OB[i])
            if (O_i == O_j == I) then
                OB.remove(i)
                OB.remove(j)
                merged ← true
            else if (O_j == I) then
                OB[i] ← O_i
                OB.remove(j)
                i ← i − 1
            else if (O_i == I) then
                OB[j] ← O_j
                OB.remove(i)
                merged ← true
            else
                OB[j] ← O_j
                OB[i] ← O_i
                LTranspose(OB, j, i)
                i ← i − 1
            end if
        end for
        if (i == 0) then
            OB̃^x.add(OB[i])
            OB.remove(i)
        end if
    end if
end while
```

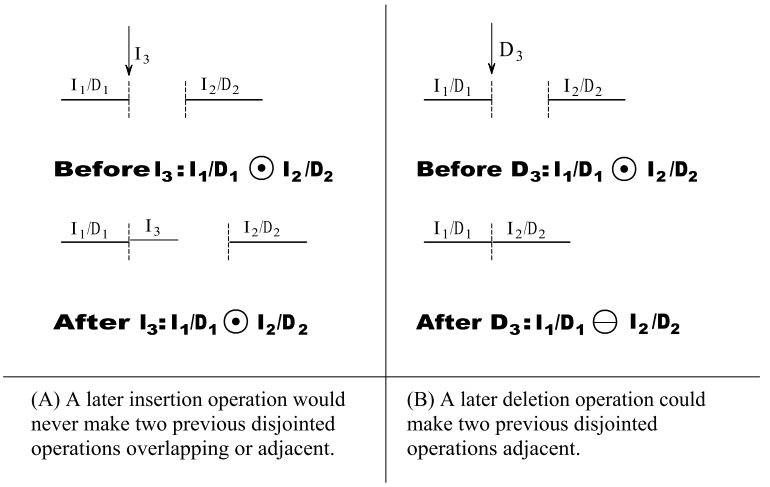
$[O_3^4, O_1^5]$, where $O_1^5 = Ins[1, 4, c123]$. This OB has achieved maximal compression, i.e., $OB = \Omega_{OB}$, achieving 60% reduction in operation number and 57% reduction in buffer size.

Second, the algorithm can discover hidden adjacent operations, i.e., these operations were initially disjointed and then made adjacent by subsequent operations. As shown in Fig. 8(A), operations I_2 (an insertion operation) or D_2 (a deletion operation) and I_1/D_1 were initially disjointed. If a new insertion operation I_3 inserts some text, for instance between the effect regions covered by I_1/D_1 and I_2/D_2, it can never make I_2/D_2 and I_1/D_1 adjacent. In contrast, a later deletion operation could create new adjacent relations among previous disjointed operations. As shown in Fig. 8(B), operations I_2/D_2 and I_1/D_1 were initially disjointed. If a new deletion operation D_3 deletes all the text between the effect regions covered by I_1/D_1 and I_2/D_2, I_2/D_2 and I_1/D_1 then becomes adjacent.

Algorithm 13 lastOp(*OB*, Type): i $(0 \leqslant i < |OB|)$

```
i ← |OB| − 1; found ← false
while (i ⩾ 0 and found == false) do
  if (T(OB[i]) == Type) then
    found ← true
  else
    i ← i − 1
  end if
end while

return i
```



(A) A later insertion operation would never make two previous disjointed operations overlapping or adjacent.

(B) A later deletion operation could make two previous disjointed operations adjacent.

Fig. 8. Hidden adjacent operations.

To address this issue, the algorithm transposes all deletion operations to the left side of all insertion operations in an *OB* in order to effectively detect the hidden adjacent relations created by these deletion operations. Therefore, the algorithm first merges all deletion operations with other operations and then merges insertion operations with the rest of the operations. In this way, insertion operations must have already taken into account the effects of all deletion operations and consequently the algorithm can merge hidden adjacent operations. As a result, a maximally compressed *OB* must look like $[D_1, \ldots, D_r, I_1, \ldots, I_s]$ where D_i $(1 \leqslant i \leqslant r)$ is a deletion operation and I_j $(1 \leqslant j \leqslant s)$ is an insertion operation.

The **COMET** (Compression by Operational Merging & Transformation) compression algorithm can achieve maximal compression. Given an $OB = [O_1, \ldots, O_n]$ storing a list of user-issued operations on a note, **COMET**(*OB*): $\Omega_{OB} = [EO_1, \ldots, EO_m]$ $(m \leqslant n)$, where EO_i $(1 \leqslant i \leqslant m)$ is referred to as an *effective operation*. A list of effective operations is the minimal list of essential operations (or operations of net effects) in transforming a note from its initial state to the final state while preserving the intentions of user-issued editing operations [25].

4.4. Verification of the compression algorithm

To verify the compression algorithm, we need to prove that the compression algorithm has achieved maximal compression (as described by Theorem 1) and the compressed buffer is equivalent to the uncompressed one in terms of transforming the note from the same initial state to the same final state (as described by Theorem 2).

Theorem 1. *Given any OB, if* **COMET***(OB) =* $\widetilde{OB}$*, then it must be* $\widetilde{OB} = \Omega_{OB}$.

Proof. For $|OB| = 1$, $\Omega_{OB} = OB$ and $\widetilde{OB} = OB$. The theorem holds.

For $|OB| = n$, hypothesize that the theorem holds, that is, given $OB = [O_1, \ldots, O_n]$, if $\widetilde{OB} = [D_1, \ldots, D_r, I_1, \ldots, I_s]$, where all D_i $(1 \leqslant i \leqslant r)$ are disjointed deletion operations, all I_j $(1 \leqslant j \leqslant s)$ are disjointed insertion operations, and $D_i \odot I_j$.

For $|OB| = n + 1$, $OB = [O_1, \ldots, O_n, O_{n+1}]$, $\widetilde{OB} =$ **COMET**$(OB) =$ **COMET**$([O_1, \ldots, O_n, O_{n+1}]) =$ **COMET**$([[O_1, \ldots, O_n], [O_{n+1}]]) =$ **COMET**$([$**COMET**$([O_1, \ldots, O_n]),$ **COMET**$([O_{n+1}])]) =$ **COMET**$([[D_1, \ldots, D_r, I_1, \ldots, I_s], [O_{n+1}]]) =$ **COMET**$([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}])$.

If O_{n+1} is an insertion operation, then for $\forall D_i$ $(1 \leqslant i \leqslant r)$, $D_i \odot O_{n+1}$. Therefore $\widetilde{OB} =$ **COMET**$([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [D_1, \ldots, D_r,$ **COMET**$([I_1, \ldots, I_s, O_{n+1}])]$.

1. Given $\forall I_j$ $(1 \leqslant j \leqslant s)$, if $I_j \odot O_{n+1}$, then $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
2. If $\exists I_k$ $(1 \leqslant k \leqslant s)$, where $I_k \oplus / \ominus O_{n+1}$, O_{n+1} will be merged into I_k. $\mathbf{COMET}([I_1, \ldots, I_k, \ldots, I_s, O_{n+1}]) = [I_1, \ldots, I'_k, \ldots, I_s]$, where all operations are disjointed. So $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [D_1, \ldots, D_r, I_1, \ldots, I'_k, \ldots, I_s]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

If O_{n+1} is a deletion operation, the proof is as follows.

1. Given $\forall D_i$ $(1 \leqslant i \leqslant r)$ or $\forall I_j$ $(1 \leqslant j \leqslant s)$, if $D_i \odot O_{n+1}$ and $I_j \odot O_{n+1}$, then $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = \mathbf{COMET}([D_1, \ldots, D_r, O'_{n+1}, I'_1, \ldots, I'_s]) = [\mathbf{COMET}([D_1, \ldots, D_r, O'_{n+1}]), \mathbf{COMET}([I'_1, \ldots, I'_s])]$. The deletion operation O_{n+1} would never create new adjacent relations among $[D_1, \ldots, D_r]$, so $\mathbf{COMET}([D_1, \ldots, D_r, O'_{n+1}]) = [D_1, \ldots, D_r, O'_{n+1}]$, where all deletion operations are disjointed. However, the deletion operation O_{n+1} may create a new adjacent relation between I_l and I_q $(1 \leqslant l, q \leqslant s)$. As a result, $\mathbf{COMET}([I'_1, \ldots, I'_s]) = [I'_1, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q+1}, \ldots, I'_s]$, where all insertion operations are disjointed. So $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [D_1, \ldots, D_r, O'_{n+1}, I'_1, \ldots, I'_s]$ or $[D_1, \ldots, D_r, O'_{n+1}, I'_1, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q+1}, \ldots, I'_s]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
2. Given $\forall I_j$ $(1 \leqslant j \leqslant s)$, where $I_j \odot O_{n+1}$, but $\exists D_k$ $(1 \leqslant k \leqslant r)$, where $D_k \oplus / \ominus O_{n+1}$, then O_{n+1} will be merged into D_k. So $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [\mathbf{COMET}([D_1, \ldots, D_r, O'_{n+1}]), \mathbf{COMET}([I'_1, \ldots, I'_s])]$. $\mathbf{COMET}([D_1, \ldots, D_r, O'_{n+1}]) = [D_1, \ldots, D'_k, \ldots, D'_r]$, where all deletion operations are disjointed. O_{n+1} may create a new adjacent relation between I_l and I_q $(1 \leqslant l, q \leqslant s)$. As a result, $\mathbf{COMET}([I'_1, \ldots, I'_s]) = [I'_1, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q+1}, \ldots, I'_s]$, where all insertion operations are disjointed. So $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [D_1, \ldots, D'_k, \ldots, D'_r, I'_1, \ldots, I'_s]$ or $[D_1, \ldots, D'_k, \ldots, D'_r, I'_1, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q+1}, \ldots, I'_s]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
3. Given $\forall D_i$ $(1 \leqslant i \leqslant r)$, where $D_i \odot O_{n+1}$ but $\exists I_k$ $(1 \leqslant k \leqslant s)$, where $I_k \oplus O_{n+1}$.
   (a) If O_{n+1} can be totally merged into I_k, $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = [D_1, \ldots, D_r, \mathbf{COMET}([I_1, \ldots, I_s, O_{n+1}])] = [D_1, \ldots, D_r, I_1, \ldots, I'_k, \ldots, I'_s]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
   (b) If I_k can be totally merged into O_{n+1}, $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = \mathbf{COMET}([D_1, \ldots, D_r, O'_{n+1}, I'_1, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I'_s]) = [D_1, \ldots, D_r, O'_{n+1}, \mathbf{COMET}([I'_1, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I'_s])]$ where D_i $(1 \leqslant i \leqslant r)$ and O'_{n+1} are disjointed deletion operations. If O_{n+1} creates a new adjacent relation between I_l and I_q $(1 \leqslant l, q \leqslant s)$, $\mathbf{COMET}([I'_1, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I'_s]) = [I'_1, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q-1}, \ldots, I'_s]$, where all insertion operations are disjointed. So $\widetilde{OB} = [D_1, \ldots, D_r, O'_{n+1}, I'_1, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I'_s]$ or $[D_1, \ldots, D_r, O'_{n+1}, I'_1, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q-1}, \ldots, I'_s]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
   (c) If I_k and O_{n+1} can be partially merged, $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}]) = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I'_k, O'_{n+1}, I'_{k+1}, \ldots, I'_s]) = \mathbf{COMET}([D_1, \ldots, D_r, O''_{n+1}, I'_1, \ldots, I''_k, \ldots, I'_s]) = [D_1, \ldots, D_r, O''_{n+1}, \mathbf{COMET}([I'_1, \ldots, I''_k, \ldots, I'_s])]$, where D_i $(1 \leqslant i \leqslant r)$ and O''_{n+1} are disjointed deletion operations. If O_{n+1} creates a new adjacent relation between I_l and I_q $(1 \leqslant l, q \leqslant s)$, $\mathbf{COMET}([I'_1, \ldots, I''_k, \ldots, I'_s]) = [I'_1, \ldots, I''_k, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q+1}, \ldots, I'_s]$, where all insertion operations are disjointed. So $\widetilde{OB} = [D_1, \ldots, D_r, O''_{n+1}, I'_1, \ldots, I''_k, \ldots, I'_s]$ or $[D_1, \ldots, D_r, O''_{n+1}, I'_1, \ldots, I''_k, \ldots, I''_l, \ldots, I'_{q-1}, I'_{q+1}, \ldots, I'_s]$, where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
4. If $\exists D_k$ $(1 \leqslant k \leqslant r)$ and I_p $(1 \leqslant p \leqslant s)$ where $D_k \oplus/\ominus O_{n+1}$ and $I_p \oplus O_{n+1}$, it can be deduced from (2) and (3) that operations in $\widetilde{OB} = \mathbf{COMET}([D_1, \ldots, D_r, I_1, \ldots, I_s, O_{n+1}])$ are also disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

By the induction argument, the theorem holds, that is, for an OB containing any number of operations, after $\mathbf{COMET}(OB)$, it must be $\widetilde{OB} = \Omega_{OB}$. □

Theorem 2. *Given any two operations O_a and O_b in a buffer, where $O_a \mapsto O_b$, if $\mathbf{OM}(O_a, O_b)$: (O'_a, O'_b), then $[O_a, O_b] \equiv [O'_a, O'_b]$.*

Proof. First reason the $\mathbf{OM\_II}(O_a, O_b)$ function. Suppose $\Upsilon_{O_a} = S$, containing a sequence of n characters $C_1 \cdots C_n$. $O_a = \mathbf{Ins}[i, r, X_1 \cdots X_r]$ $(0 \leqslant i \leqslant n)$ is to insert a sequence of r characters $X_1 \cdots X_r$ at position i.

1. If $O_a \odot O_b$, then $\mathbf{OM\_II}(O_a, O_b)$: $(O'_a, O'_b) = (O_a, O_b)$. As a result, $S \circ [O_a, O_b] = S \circ \mathbf{OM\_II}(O_a, O_b) = S \circ [O'_a, O'_b]$. So $[O_a, O_b] \equiv [O'_a, O'_b]$ holds.
2. If $O_a \ominus O_b$, then it must be $O_b = \mathbf{Ins}[i, s, Y_1 \cdots Y_s]$ to insert a sequence of s characters $Y_1 \cdots Y_s$ just left to the string inserted by O_a, or $O_b = \mathbf{Ins}[i+r, s, Y_1 \cdots Y_s]$ to insert a sequence of s characters $Y_1 \cdots Y_s$ just right to the string inserted by O_a. As a result, $\mathbf{OM\_II}(O_a, O_b)$: (O'_a, I) where $O'_a = \mathbf{Ins}[i, s+r, Y_1 \cdots Y_s X_1 \cdots X_r]$ or $\mathbf{Ins}[i, r+s, X_1 \cdots X_r Y_1 \cdots Y_s]$.
   - $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_i Y_1 \cdots Y_s X_1 \cdots X_r C_{i+1} \cdots C_n$ or $C_1 \cdots C_i X_1 \cdots X_r Y_1 \cdots Y_s C_{i+1} \cdots C_n$.

- $S \circ [O'_a] = C_1 \cdots C_i Y_1 \cdots Y_s X_1 \cdots X_r C_{i+1} \cdots C_n$ or $C_1 \cdots C_i X_1 \cdots X_r Y_1 \cdots Y_s C_{i+1} \cdots C_n$.
- As a result, $S \circ [O_a, O_b] = S \circ [O'_a] = S \circ \mathbf{OM\_II}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a]$ holds.

3. If $O_a \oplus O_b$, then it must be $O_b = \mathbf{Ins}[j, s, Y_1 \cdots Y_s]$ where $i < j < i + r$ to insert a sequence of s characters within the string inserted by O_a. As a result, $\mathbf{OM\_II}(O_a, O_b)$: (O'_a, I) where $O'_a = \mathbf{Ins}[i, r + s, X_1 \cdots X_{j-i} Y_1 \cdots Y_s X_{j-i+1} \cdots X_r]$.
   - $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_i X_1 \cdots X_{j-i} Y_1 \cdots Y_s X_{j-i+1} \cdots X_r C_{i+1} \cdots C_n$.
   - $S \circ [O'_a] = C_1 \cdots C_i X_1 \cdots X_{j-i} Y_1 \cdots Y_s X_{j-i+1} \cdots X_r C_{i+1} \cdots C_n$.
   - As a result, $S \circ [O_a, O_b] = S \circ [O'_a] = S \circ \mathbf{OM\_II}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a]$ holds.

Then reason the $\mathbf{OM\_DD}(O_a, O_b)$ function. Suppose $\Upsilon_{O_a} = S$, containing a sequence of n characters $C_1 \cdots C_n$. $O_a = \mathbf{Del}[i, r, C_{i+1} \cdots C_{i+r}]$ $(0 \leqslant i \leqslant n)$ is to delete a sequence of r characters $C_{i+1} \cdots C_{i+r}$ at position i.

1. If $O_a \odot O_b$, then $\mathbf{OM\_DD}(O_a, O_b)$: $(O'_a, O'_b) = (O_a, O_b)$. As a result, $S \circ [O_a, O_b] = S \circ \mathbf{OM\_DD}(O_a, O_b) = S \circ [O'_a, O'_b]$. So $[O_a, O_b] \equiv [O'_a, O'_b]$ holds.
2. If $O_a \ominus O_b$, then it must be $O_b = \mathbf{Del}[j, i - j, C_{j+1} \cdots C_i]$ where $j < i$ to delete a sequence of $i - j$ characters just left to the string deleted by O_a at position j, or $O_b = \mathbf{Del}[i, s, C_{i+r+1} \cdots C_{i+r+s}]$ to delete a sequence of s characters just right to the string deleted by O_a at position i. As a result, $\mathbf{OM\_DD}(O_a, O_b)$: (O'_a, I) where $O'_a = \mathbf{Del}[j, i - j + r, C_{j+1} \cdots C_{i+r}]$ or $\mathbf{Del}[i, r + s, C_{i+1} \cdots C_{i+r+s}]$.
   - $S_a = S \circ [O_a] = C_1 \cdots C_i C_{i+r+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_j C_{i+r+1} \cdots C_n$ or $C_1 \cdots C_i C_{i+r+s+1} \cdots C_n$.
   - $S \circ [O'_a] = C_1 \cdots C_j C_{i+r+1} \cdots C_n$ or $C_1 \cdots C_i C_{i+r+s+1} \cdots C_n$.
   - As a result, $S \circ [O_a, O_b] = S \circ [O'_a] = S \circ \mathbf{OM\_DD}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a]$ holds.
3. If $O_a \oplus O_b$, then it must be $O_b = \mathbf{Del}[j, s, C_{j+1} \cdots C_i C_{i+r+1} \cdots C_{r+s+j}]$ where $j < i$ and $s > i - j$. As a result, $\mathbf{OM\_DD}(O_a, O_b) = (O'_a, I)$ where $O'_a = \mathbf{Del}[j, r + s, C_1 \cdots C_j C_{r+s+j+1} \cdots C_n]$.
   - $S_a = S \circ [O_a] = C_1 \cdots C_i C_{i+r+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_j C_{r+s+j+1} \cdots C_n$.
   - $S \circ [O'_a] = C_1 \cdots C_j C_{r+s+j+1} \cdots C_n$.
   - As a result, $S \circ [O_a, O_b] = S \circ [O'_a] = S \circ \mathbf{OM\_DD}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a]$ holds.

Finally reason the $\mathbf{OM\_ID}(O_a, O_b)$ function. Suppose $\Upsilon_{O_a} = S$, containing a sequence of n characters $C_1 \cdots C_n$. $O_a = \mathbf{Ins}[i, r, X_1 \cdots X_r]$ $(0 \leqslant i \leqslant n)$ is to insert a sequence of r characters $X_1 \cdots X_r$ at position i.

1. If $O_a \odot O_b$, then $\mathbf{OM\_ID}(O_a, O_b)$: $(O'_a, O'_b) = (O_a, O_b)$. As a result, $S \circ [O_a, O_b] = S \circ \mathbf{OM\_ID}(O_a, O_b) = S \circ [O'_a, O'_b]$. So $[O_a, O_b] \equiv [O'_a, O'_b]$ holds.
2. $O_a \oplus O_b$, then it must be one of the following possibilities:
   (a) $O_b = \mathbf{Del}[i, r, X_1 \cdots X_r]$ to delete the r characters $X_1 \cdots X_r$ inserted by O_a at position i. As a result, $\mathbf{OM\_ID}(O_a, O_b) = [I, I]$.
      - $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_i C_{i+1} \cdots C_n = C_1 \cdots C_n = S$.
      - $S \circ \mathbf{OM\_ID}(O_a, O_b) = S \circ [\,] = S$.
      - As a result, $S \circ [O_a, O_b] = S \circ [\,] = S \circ \mathbf{OM\_ID}(O_a, O_b)$. So $[O_a, O_b] \equiv [\,]$ holds.
   (b) $O_b = \mathbf{Del}[j, s, C_{j+1} \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_{s+j-r}]$ where $j < i$ and $s > i - j + r$ to delete a sequence of $i - j$ characters left to the string inserted by O_a, the entire string $X_1 \cdots X_r$ inserted by O_a, and a sequence of $s + j - i - r$ characters right to the string inserted by O_a. As a result, $\mathbf{OM\_ID}(O_a, O_b)$: $[I, O'_b]$ where $O'_b = \mathbf{Del}[j, s - r, C_{j+1} \cdots C_{s+j-r}]$.
      - $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_j C_{s+j-r+1} \cdots C_n$.
      - $S \circ \mathbf{OM\_ID}(O_a, O_b) = S \circ [O'_b] = C_1 \cdots C_j C_{s+j-r+1} \cdots C_n$.
      - As a result, $S \circ [O_a, O_b] = S \circ O'_b] = S \circ \mathbf{OM\_ID}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_b]$ holds.
   (c) $O_b = \mathbf{Del}[j, s, X_{j-i+1} \cdots X_{s+j-i}]$ where $i < j < i + r$ and $s < r + i - j$ to delete part of the string inserted by O_a. As a result, $\mathbf{OM\_ID}(O_a, O_b)$: $[O'_a, I]$ where $O'_a = \mathbf{Ins}[i, r - s, X_1 \cdots X_{j-i} X_{s+j-i+1} \cdots X_r]$.
      - $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_i X_1 \cdots X_{j-i} X_{s+j-i+1} \cdots X_r C_{i+1} \cdots C_n$.
      - $S \circ \mathbf{OM\_ID}(O_a, O_b) = S \circ [O'_a] = C_1 \cdots C_i X_1 \cdots X_{j-i} X_{s+j-i+1} \cdots X_r C_{i+1} \cdots C_n$.
      - As a result, $S \circ [O_a, O_b] = S \circ S[O'_a] = \circ \mathbf{OM\_ID}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a]$ holds.
   (d) $O_b = \mathbf{Del}[j, s, C_{j+1} \cdots C_i X_1 \cdots X_{s+j-i}]$ where $j < i$ and $i - j < s < r + i - j$ to delete a sequence of $i - j$ characters left to the string inserted by O_a and a left part of the string inserted by O_a. As a result, $\mathbf{OM\_ID}(O_a, O_b)$: $[O'_a, O'_b]$ where $O'_a = \mathbf{Ins}[i, r - s + i - j, X_{s+j-i+1} \cdots X_r]$ and $O'_b = \mathbf{Del}[j, i - j, C_{j+1} \cdots C_i]$.
      - $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_j X_{s+j-i+1} \cdots X_r C_{i+1} \cdots C_n$.
      - $S'_a = S \circ [O'_a] = C_1 \cdots C_i X_{s+j-i+1} \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O'_a, O'_b] = S'_a \circ [O'_b] = C_1 \cdots C_j X_{s+j-i+1} \cdots X_r C_{i+1} \cdots C_n$.
      - As a result, $S \circ [O_a, O_b] = S \circ [O'_a, O'_b] = S \circ \mathbf{OM\_ID}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a, O'_b]$ holds.
   (e) $O_b = \mathbf{Del}[j, s, X_{j-i+1} \cdots X_r C_{i+1} \cdots C_{s+j-r}]$, where $i < j \leqslant r + i$ and $s > r + i - j$ to delete a right part of the string inserted by O_a and a sequence of $s - r + j - i$ characters right to the string inserted by O_a. As a result, $\mathbf{OM\_ID}(O_a, O_b)$: $[O'_a, O'_b]$ where $O'_a = \mathbf{Ins}[i, j - i, X_1 \cdots X_{j-i}]$ and $O'_b = \mathbf{Del}[j, s - r + j - i, C_{i+1} \cdots C_{s+j-r}]$.

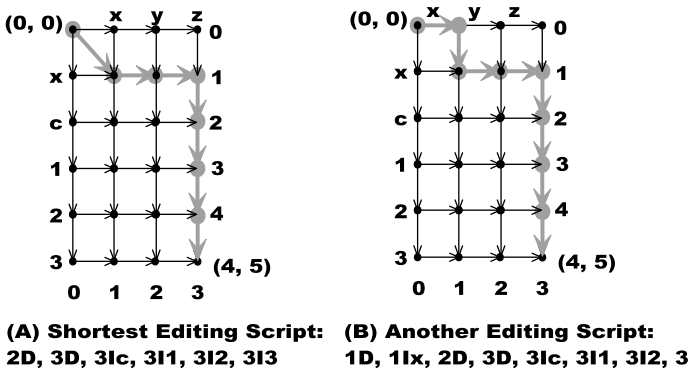


Fig. 9. Editing graph transforming string *xyz* to string *xc*123.

- $S_a = S \circ [O_a] = C_1 \cdots C_i X_1 \cdots X_r C_{i+1} \cdots C_n$ and $S \circ [O_a, O_b] = S_a \circ [O_b] = C_1 \cdots C_i X_1 \cdots X_{j-i} C_{s+j-r+1} \cdots C_n$.
- $S'_a = S \circ [O'_a] = C_1 \cdots C_i X_+ 1 \cdots X_{j-i} C_{i+1} \cdots C_n$ and $S \circ [O'_a, O'_b] = S'_a \circ [O'_b] = C_1 \cdots C_i X_1 \cdots X_{j-i} C_{s+j-r+1} \cdots C_n$.
- As a result, $S \circ [O_a, O_b] = S \circ [O'_a, O'_b] = S \circ \mathbf{OM\_ID}(O_a, O_b)$. So $[O_a, O_b] \equiv [O'_a, O'_b]$ holds. □

A list of effective operations is equivalent to a list of editing scripts derived by text differentiation algorithms [14]. A list of effective operations is one alternative list of editing scripts used to transform a note from its initial state to its final state, while a list of editing scripts derived by text differentiation algorithms is the shortest list of editing scripts used to transform it from the same initial state to the same final state. The difference between the two lists is that the former list preserves the intentions of user-issued actions while the latter list attempts to reconstruct actions after the fact and has little chance to preserve the intentions of user-issued actions. For the sake of comparison, both effective operations and editing scripts are represented as character-based.

For the example shown in Fig. 6, when *OB* is compressed, $\Omega_{OB} = \mathbf{COMET}(OB)$: $[O_3^4,\ O_1^2]$ where $O_3^4 = \mathbf{Del}[1, 2, yz]$ and $O_1^2 = \mathbf{Ins}[1, 4, c123]$ are effective operations. If effective operations are represented as character-based, then $\Omega_{OB} = [EO_1, EO_2, EO_3, EO_4, EO_5, EO_6]$ where $EO_1 = \mathbf{Del}[1,\ 1,\ y]$, $EO_2 = \mathbf{Del}[1, 1, z]$, $EO_3 = \mathbf{Ins}[1, 1, c]$, $EO_4 = \mathbf{Ins}[2, 1, 1]$, $EO_5 = \mathbf{Ins}[3, 1, 2]$, and $EO_6 = \mathbf{Ins}[4, 1, 3]$. The editing graph for transforming string *xyz* to string *xc123* is shown in Fig. 9.

According to the text differentiation algorithm [14], the shortest editing script for transforming string *xyz* to string *xc123* contains five editing operations shown in Fig. 9(A): **2D**(delete character *y*), **3D**(delete character *z*), **3Ic**(insert character *c*), **3I1**(insert character 1), **3I2**(insert character 2), and **3I3**(insert character 3). In this example, the list of shortest editing scripts derived by the text differentiation algorithm accidentally coincides with the list of effective operations $[EO_1, EO_2, EO_3, EO_4, EO_5, EO_6]$.

As pointed out, the list of effective operations and the list of shortest editing scripts are two of many alternative paths in transforming a note from its initial state to the final state. These two paths may not necessarily be the same because the user may not necessarily choose the shortest path to transform the note from its initial state to the final state. For example, in Fig. 9(B), the user may choose another path that is different from the shortest path in Fig. 9(A) to transform string *xyz* to string *xc123*. That path consists of the following list of effective operations: **1D**(delete character *x*), **1Ix**(insert character *x*), **2D**(delete character *y*), **3D**(delete character *z*), **3Ic**(insert character *c*), **3I1**(insert character 1), **3I2**(insert character 2), and **3I3**(insert character 3). Nevertheless, the scale of effective operations is comparable to that of the shortest list of editing scripts derived by text differentiation algorithms in terms of both the size of the list and the number of operations within the list.

5. Personalized content retrieval

In *GroupNotes*, each user owns a note, which is viewable and editable by members in a real-time collaborative note-taking session. A community note that combines notes from all members in the session is automatically generated by the *GroupNotes* server and made available to each member after the session is over.

A user can request to view a live note from a real-time collaborative note-taking session where the user is not a member by retrieving it with the session identity and the note identity within the session, for example ⟨*session K* : *note i*⟩. If the request is granted by the note owner, the user can view it while it is being updated in real time and add it to the communicate note that will become available to the user after the session is over. A live note can also be retrieved through other search criteria such as the owner's identity or the note's unique identity.

Constrained by the mobile device's screen real estate and the human's cognitive power, a user can only retrieve a few live notes from other real-time collaborative note-taking sessions, however a user may want to retrieve significant number of saved notes to generate a comprehensive community note or facilitate social learning through rating (e.g. using a score from 0 to 5), annotating (e.g. using pedagogical or topic tags), following (e.g., commenting, clarifying, or questioning) another user's note. All notes are saved in the *GroupNotes* server in a hierarchical structure shown in Fig. 10.

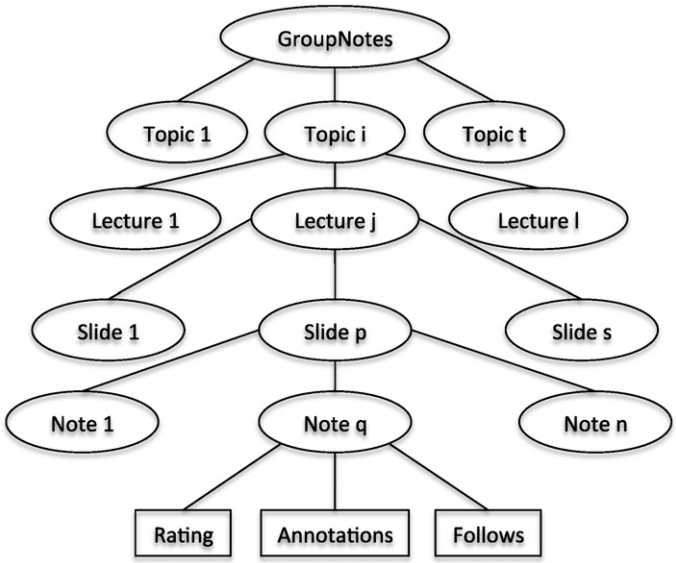


Fig. 10. The hierarchical structure of saved notes.

A saved note can be uniquely identified by the topic identity, the lecture identity within the topic, the slide identity within the lecture, and the note identity within the slide, for example $\langle Topic\ i : Lecture\ j : Slide\ p : note\ q \rangle$. Saved notes can be retrieved in the following customizable search methods. For the sake of simplicity without losing generality, it is assumed that the topic identity, the lecture identity within the topic, and the slide identity within the lecture are all known, that is, a user wants to retrieve saved notes for a given slide.

1. Precisely retrieve a note using the unique note identity or the note owner's unique identity.
2. Retrieve and rank notes according to their ratings.
3. Retrieve and rank notes according to the number of follows.
4. Retrieve notes using a specified tag or a combination of diverse tags.
5. Retrieve notes that (or their follows) match provided query terms.
6. An arbitrary combination of methods 2–5.

It is worth clarifying that retrieval of saved notes is a post real-time collaborative note-taking activity, but it is an essential activity that disseminates the notes produced in collaborative sessions for facilitating further social learning that may eventually lead to the improvement of learning outcomes. It represents a mobile social software application that can transit from real-time collaboration mode to non-real-time collaboration mode smoothly and naturally.

6. Conclusions and future work

Current computing technology is being driven by two major forces: mobile devices and social software. Mobile devices allow people to have access to information anytime anywhere even when they are on the go and social software enables people to network without being physically present. The convergence of these two forces is mobile social software, currently exemplified by non-real-time email systems and social media systems on mobile handheld devices.

Real-time social software, which has been rapidly growing since the advent of *Google Docs*, is virtually non-existent on mobile handheld devices (except real-time chat and instant messaging). The device characteristics such as small screen real estate, limited battery talk time, scarce network resources, and inherent need for personalization present challenges to the design and implementation of effective and useful real-time social software. In this article, we have presented a technical solution to these challenges using a smartphone-based real-time collaborative note-taking system as an example.

The solution allows for personalized multi-user views through flexible layout of multiple windows, maximally utilizing the available screen real estate, personalized content synchronization through OT-based synchronization protocols and algorithms and an OM-based buffer compression algorithm, maximally utilizing the available battery talk time and network resources, and personalized content retrieval through customizable search methods.

Imminent future work includes the validation of the proposed solution by collecting and analyzing user data pertaining to the consumption of battery talk time and network resources and the evaluation of the multi-user interface in relation to the screen real estate and the usability of the smartphone-based real-time collaborative note-taking system.

Acknowledgments

The work presented in this article was partially supported by an establishment grant (36426) from Flinders University, Australia. The authors would like to thank Lijun Cao and Mou Chen for their contributions to the development of the *GroupNotes* apps running on Android smartphone and tablets.

References

[1] J. Begole, M.B. Rosson, C.A. Shaffer, Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems, ACM Trans. Comput.-Hum. Interact. 6 (2) (June 1999) 95–132.
[2] P. Bernstein, N. Goodman, V. Hadzilacos, Concurrency Control and Recorvery in Database Systems, Addison–Wesley, 1987.
[3] E.A. Bier, S. Freeman, MMM: a user interface architecture for shared editors on a single screen, in: Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology, October 1991, pp. 79–86.
[4] T. Blechschmidt, T. Wieland, C. Kuhmunch, L. Mehrmann, Personalization of end user software on mobile devices, in: The 2nd IEEE International Workshop on Mobile Commerce and Services, 2005, pp. 130–137.
[5] S. Counts, H. ter Hofte, I. Smith, Mobile social software: realizing potential, managing risks, in: CHI'06 Extended Abstracts on Human Factors in Computing, 2006, pp. 1703–1706.
[6] C. Ellis, S. Gibbs, G. Rein, Groupware: some issues and experiences, Commun. ACM 34 (1) (January 1991) 39–58.
[7] S. Greenberg, C. Gutwin, A. Cockburn, Awareness through fisheye views in relaxed-WYSIWIS groupware, in: Proc. of Graphics Interface, Morgan Kaufmann, May 1996, pp. 28–38.
[8] S. Greenberg, D. Marwood, Real time groupware as a distributed system: concurrency control and its effect on the interface, in: Proceedings of ACM Conference on Computer Supported Cooperative Work, November 1994, pp. 207–217.
[9] A. Karsenty, M. Beaudouin-Lafon, An algorithm for distributed groupware applications, in: Proceedings of the 13th Conference on Distributed Groupware Computing Systems, May 1993, pp. 195–202.
[10] M. Knister, A. Prakash, Issues in the design of a toolkit for supporting multiple group editors, J. Usenix Assoc. 6 (2) (1993) 135–166.
[11] E. Lippe, N.V. Oosterom, Operation-based merging, in: Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments, November 1992, pp. 78–87.
[12] J. Manweiler, S. Agarwal, M. Zhang, R.R. Choudhury, P. Bahl, Switchboard: a matchmaking system for multiplayer mobile games, in: The 9th International Conference on Mobile Systems, Applications, and Services, 2011, pp. 71–84.
[13] J.P. Munson, P. Dewan, A flexible object merging framework, in: Proceedings of ACM Conference on Computer-Supported Cooperative Work, ACM Press, October 1994, pp. 231–242.
[14] E. Myers, An O(ND) difference algorithm and its variations, Algorithmica 1 (2) (1986) 251–266.
[15] K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer, A.J. Demers, Flexible update propagation for weakly consistent replication, in: Proc. of the Sixteenth ACM Symposium on Operating Systems Principles, October 1997, pp. 288–301.
[16] M. Raynal, M. Singhal, Logical time: capturing causality in distributed systems, IEEE Comput. Mag. 29 (2) (Feb. 1996) 49–56.
[17] M. Reilly, H. Shen, Groupnotes: encouraging proactive student engagement in lectures through collaborative note-taking on smartphones, in: The 9th International Conference on Computer Supported Collaborative Learning, 2011, pp. 908–909.
[18] M. Reilly, H. Shen, Shared note-taking: a smartphone-based approach to increased student engagement in lectures, in: The 11th International Workshop on Collaborative Editing Systems in Conjunction with ACM Conference on Computer Supported Cooperative Work, 2011.
[19] H. Shen, C. Sun, A log compression algorithm for operation-based version control systems, in: Proceedings of IEEE 26th Annual International Computer Software and Application Conference, August 2002, pp. 867–872.
[20] H. Shen, C. Sun, Flexible notification for collaborative systems, in: ACM Conference on Computer Supported Cooperative Work, 2002, pp. 77–86.
[21] H. Shen, C. Sun, Achieving data consistency by contextualization in collaborative web-based applications, ACM Trans. Internet Technol. 10 (4) (2011) 1–37.
[22] H. Shen, S. Zhou, C. Sun, Z. Phyo, A generic WebDAV-based document repository manager for collaborative systems, in: Proceedings of the 2006 IEEE/WIC/ACM Conference on Web Intelligence, December 2006, pp. 129–136.
[23] M. Stefik, D.F. Bobrow, G. Foster, S. Lanning, D. Tatar, WYSIWIS revised: early experiences with multiuser interfaces, ACM Trans. Inform. Syst. 5 (2) (1987) 147–167.
[24] C. Sun, C. Ellis, Operational transformation in real-time group editors: issues, algorithms, and achievements, in: ACM Conference on Computer Supported Cooperative Work, 1998, pp. 59–68.
[25] C. Sun, X. Jia, Y. Zhang, Y. Yang, D. Chen, Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems, ACM Trans. Comput.-Hum. Interact. 5 (1) (1998) 63–108.
[26] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, W. Cai, Transparent adaptation of single-user applications for multi-user real-time collaboration, ACM Trans. Comput.-Hum. Interact. 13 (4) (December 2006) 531–582.
[27] D. Sun, S. Xia, C. Sun, D. Chen, Operational transformation for collaborative word processing, in: Proceedings of ACM Conference on Computer Supported Cooperative Work, November 2004, pp. 437–446.
[28] The EDUCAUSE Learning Initiative, 7 things you should know about collaborative editing, in http://www.educause.edu/ELI/7ThingsYouShouldKnowAboutColla/156812, December 2005.
[29] S. Xia, D. Sun, C. Sun, D. Chen, An integrated session and repository management approach for real-time collaborative editing systems, in: The 4th International Conference on Creating, Connecting and Collaborating through Computing, 2006, pp. 254–261.
[30] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inform. Theory 23 (3) (1977) 337–343.